INTRODUCTION :

VISUAL BASIC, UN LANGAGE (PRESQUE) OBJET

Foin de fausse modestie, ce cours poursuit un double objectif :

- constituer un vade-mecum de départ pour le langage Visual Basic, dans sa version 5 (mais cela vaut tout aussi bien pour la version 6).
- présenter certains des concepts fondamentaux de la programmation objet

Visual Basic étant, comme tout langage moderne, richissime en fonctionnalités, il va de soi que les quelques pages qui suivent ne remplaceront ni une consultation intelligente de l'aide du logiciel, ni le recours à des ouvrages de référence d'une toute autre ampleur (mais d'un tout autre prix. On n'a rien sans rien). En revanche, elles visent à brosser à grands traits les spécificités de Visual Basic, posant ainsi les fondations d'un apprentissage technique plus approfondi.

Ajoutons que ce cours serait vide de sens sans les exercices – et les corrigés – qui l'accompagnent.

Merci de votre attention, vous pouvez reprendre votre sieste.

1. PARTICULARITES DES LANGAGES OBJET

En quoi un langage objet diffère-t-il d'un langage normal ? On peut résumer l'affaire en disant qu'un langage objet possède toutes les caractéristiques d'un langage traditionnel, avec deux grands aspects supplémentaires.

Donc, c'est un premier point, on peut tout à fait programmer dans un langage objet comme on programmerait du Fortran, du Cobol ou du C. Selon le vieil adage, qui peut le plus peut le moins. En pratique, cela voudrait dire négliger tout ce qui fait la spécificité d'un tel langage, comme - entre autres - la prise en charge de l'environnement graphique Windows.

Cela implique également que toutes les notions fondamentales que le programmeur a mises en pratique en algorithmique ou en programmation dans un langage traditionnel conservent leur validité pleine et entière : comme tout langage, un langage objet ne connaît que quatre grands types d'instructions : affectations de variables, tests, boucles et entrées / sorties (encore que, nous le verrons, ce dernier type puisse y connaître de fait un certain nombre de bouleversements). Comme tout langage, un langage objet connaît des variables de différents types (numérique, caractère, booléen), et des variables indicées (tableaux). Donc, encore une fois, tout ce qui était vrai dans la programmation traditionnelle demeure vrai dans la programmation objet.

Mais celle-ci offre comme on vient de le dire deux nouveaux outils, redoutables de puissance, à la trousse du programmeur.

1.1 Les Objets

1.1.1 Présentation

La première particularité d'un langage objet est de mettre à votre disposition des objets. Etonnant, non ?

Un objet peut être considéré comme une structure supplémentaire d'information, une espèce de super-variable. En effet, nous savons qu'une **variable** est un emplacement en mémoire vive,

caractérisé par une adresse – un nom – et un type (entier, réel, caractère, booléen, etc.). Dans une variable, on ne peut stocker qu'une information et une seule. Même dans le cas où l'on emploie une variable indicée – un tableau – les différents emplacements mémoire ainsi définis stockeront tous obligatoirement des informations de même type.

Un objet est un groupe de variables de différents types. Il rassemble ainsi couramment des dizaines d'informations très différentes les unes des autres au sein d'une même structure, rendant ainsi ces informations plus faciles à manier.

A la différence de ce qui se passe avec un tableau, les différentes variables d'un même objet ne sont pas désignées par un indice, mais par un nom qui leur est propre. En l'occurrence, ces noms qui caractérisent les différentes variables au sein d'un objet s'appellent des **propriétés de l'objet**. Conséquence, **toute propriété d'objet obéit strictement aux règles qui s'appliquent aux variables dans tout langage (type, taille, règles d'affectation…).**

On dira également que plusieurs objets qui possèdent les mêmes propriétés sont du même type, ou encore pour mieux frimer, de la même **classe**. Clââââsse !

A titre d'exemple, prenons un objet d'usage courant : un ministre.

Les **propriétés** d'un ministre sont : sa taille, son poids, son âge, son portefeuille, le montant de son compte en Suisse, son nom, sa situation par rapport à la justice, etc.

On peut retrouver aisément le type de chacune de ces propriétés :

- le portefeuille, le nom, sont des propriétés de type caractère.
- la taille, le poids, l'âge, le compte en Suisse, sont des propriétés de type numérique.
- la situation judiciaire (mis en examen ou non) est une propriété booléenne.

1.1.2 Syntaxe

La syntaxe qui permet de désigner une propriété d'un objet est :

```
objet.propriété
```

Par exemple, nous pouvons décider que le montant du compte en Suisse du ministre Duchemol s'élève modestement à 100 000 euros. Si la propriété désignant ce compte pour les objets de type (de classe) ministre est la propriété CompteSuisse, on écrira donc l'instruction suivante :

```
Duchemol.CompteSuisse = 100 000
```

Pour affecter à la variable Toto le montant actuel du compte en Suisse du ministre Duchemol, on écrira :

```
Toto = Duchemol.CompteSuisse
```

Pour augmenter de 10 000 euros le montant du compte en Suisse de Duchemol, on écrira :

```
Duchemol.CompteSuisse = Duchemol.CompteSuisse + 10  000
```

Et, vraiment juste histoire d'utiliser une propriété booléenne, et parce que Duchemol n'est pas le seul objet de la classe ministre :

```
Pasqua.MisEnExamen = True
```

On répète donc qu'hormis ce qui concerne la syntaxe, l'usage des propriétés des objets ne se différencie en rien de celui des variables classiques.

1.1.3 Méthodes

Les langages objet ont intégré une autre manière d'agir sur les objets : les **méthodes**.

Une méthode est une action sur l'une – ou plusieurs - des propriétés d'un objet.

Une méthode va supposer l'emploi d'un certain nombre d'**arguments**, tout comme une fonction. On trouvera donc des méthodes à un argument, des méthodes à deux arguments (plus rares), et aussi des méthodes sans arguments.

Ce n'est pas le seul point commun entre les méthodes et les fonctions.

On sait qu'une fonction peut : soit accomplir une tâche impossible si elle n'existait pas, soit accomplir une tâche possible par d'autres moyens, mais pénible à mettre en œuvre.

De la même manière, certaines méthodes accomplissent des tâches qui leur sont propres, et qui ne pourraient pas être accomplies si elles n'existaient pas. D'autres méthodes ne sont là que pour soulager le programmeur, en permettant de modifier rapidement un certain nombre de propriétés.

Par exemple, reprenons le cas notre ministre. Une méthode pourrait être **AugmenterPatrimoine**, qui supposerait un argument de type numérique. On pourrait ainsi écrire :

Duchemol.AugmenterPatrimoine(10 000)

Ce qui aurait en l'occurrence exactement le même effet que de passer par la propriété correspondante :

```
Duchemol.CompteSuisse = Duchemol.CompteSuis se + 10 000
```

1.1.4 Conclusion

Pour terminer sur ce sujet, il faut bien faire attention à une chose lorsqu'on utilise des objets.

- certains objets sont fournis par le langage de programmation lui-même. Il s'agit en particulier (mais pas seulement) de ce qu'on appelle des contrôles, c'est-à-dire d'objets possédant pour la plupart une existence graphique ; ce sont des **éléments de l'interface Windows**. Pour tous ces objets que le programmeur utilise alors qu'ils ont été créés par d'autres, les propriétés et les méthodes ne s'inventent pas : chaque type (chaque classe) d'objet possède ses propres méthodes et arguments, qu'il s'agit donc de connaître pour utiliser l'objet en question. Quitte à insister, je répète : connaître, et non inventer.
- d'autre part, un langage objet ouvre la possibilité de créer soi-même ses propres objets, et donc de programmer leurs propriétés et leurs méthodes. On se situe alors à tout autre niveau, celui de la **programmation objet** proprement dite. Nous n'aborderons ce domaine que pour mémoire, tout à la fin du cours. Ne soyons pas prétentieux, et commençons par le commencement.

En attendant, l'essentiel de nos efforts va consister à comprendre comment on peut se servir des objets (en particulier des contrôles) écrits par d'autres. Et pour ceux qui trouveraient - ils auraient raison - que ces quelques lignes sont loi d'épuiser le sujet, je leur donne rendez-vous au dernier chapitre de ce cours.

1.2 Procédures événementielles

On en arrive à la deuxième grande possibilité supplémentaire des langages objet par rapport aux langages traditionnels.

En PASCAL ou en C, par exemple, une application est constituée d'une procédure principale contenant la totalité du code (y compris par l'appel indirect à des sous-programmes). Les instructions qu'elle contient sont exécutées les unes après les autres, jusqu'à la fin (je passe pudiquement sous silence l'improbable hypothèse d'un arrêt prématuré pour cause d'erreur).

Le point fondamental est que dans un tel langage, l'ordre d'exécution des procédures et des sous-procédures est entièrement fixé d'avance par le programmeur lui-même, par le biais des instructions d'appel des sous-procédures. Par ailleurs, et ce n'est pas un hasard, ces procédures portent des noms arbitraires fixés par le programmeur, hormis le cas particulier de la procédure principale, qui se doit de porter un nom particulier, fixé par le langage (généralement, Main).

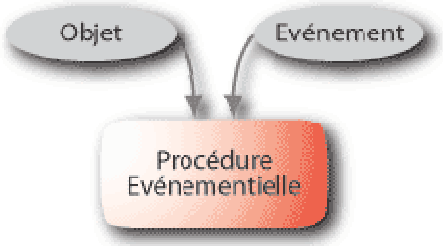
Dans un langage objet, on peut, si on le désire, conserver intégralement ce mode de fonctionnement. Mais ce n'est plus le seul possible.

En effet, dans un langage objet, il n'y a donc plus à proprement parler de procédure principale; en tout cas, l'existence d'une procédure principale n'a rien d'obligatoire.

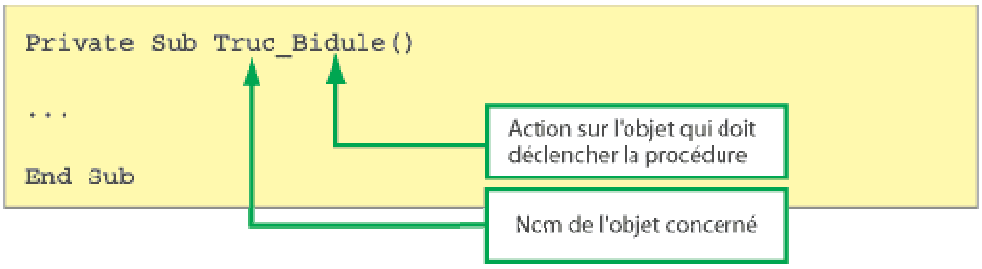
Chaque procédure est liée à la survenue d'un **événement** sur un objet, et sera donc automatiquement exécutée lorsque cet événement se produit. **Le nom de la procédure est alors, de manière obligatoire, le nom de la combinaison objet-événement qui la déclenche**.

- On vient de parler des **objets**, et en particulier des **contrôles**. Répétons qu'un contrôle est un des éléments de l'interface graphique de Windows, éléments que VB met à la disposition du programmeur pour qu'il constitue ses propres applications. Ainsi, les contrôles les plus fréquents sont : la feuille, le bouton de commande, la liste, la case à cocher, le bouton radio, etc.
- Quant aux **événements,** ils peuvent être déclenchés par l'utilisateur, ou par déroulement du programme lui-même. Les événements déclenchés par l'utilisateur sont typiquement : la frappe au clavier, le clic, le double-clic, le cliquer-glisser. Les événements déclenchés par le code sont des instructions qui modifient, lors de leur exécution, une caractéristique de l'objet ; par exemple, le redimensionnement, le déplacement, etc.

Résumons-nous. Un petit dessin vaut parfois mieux qu'un grand discours :



Le lien entre l'objet, la survenue de l'événement et le déclenchement de la procédure est établi par le nom de la procédure lui-même. Ainsi, le nom d'une procédure événementielle répond à une syntaxe très précise :



Par exemple, la procédure suivante :

```
Private Sub Machin_Click()
…
End Sub
```

Se déclenche si et seulement si l'utilisateur clique sur l'objet dont le nom est "Machin". Si on prend le problème dans l'autre sens : si je suis programmeur, et que je veux qu'il se passe ceci et cela lorsque l'utilisateur clique sur le bouton appelé "Go", je dois créer une procédure qui s'appellera obligatoirement **Sub Go_Click()** et qui contiendra les instructions "ceci" et "cela".

Moralité : Ecrire un programme qui exploite l'interface Windows, c'est avant tout commencer par définir :

- quels sont les objets qui figureront à l'écran
- ce qui doit se passer lorsque l'utilisateur agit sur ces objets via la souris ou le clavier.

A la différence d'un programme traditionnel, les procédures liées aux différents événements possibles ne seront donc pas toujours exécutées dans le même ordre: tout dépend de ce que fera l'utilisateur.

Mais bien sûr, à l'intérieur de chaque procédure, les règles traditionnelles de programmation restent vraies. C'est déjà ça de pris.

Si vous avez bien compris ce qui précède, vous avez accompli 80 % de la tâche qui vous attend en ce second semestre. Non, ce n'est pas de la démagogie. J'ai bien dit : « si vous avez bien compris ».

haut de page

2. Compilation et Interpretation

Lorsqu'on écrit une application Visual Basic (on ne traite ici que du cas standard, il en existe d'autres, mais qui sortent du sujet du cours), on crée donc un ensemble d'objets à partir des classes proposées, et on définit les procédures qui se rapportent à ces objets. Lorsqu'on sauvegarde cette application, Visual Basic va créer un certain nombre de fichiers. Pour l'essentiel :

- un fichier dit **Projet** comportant l'extension ***.vbp** (les plus fûtés d'entre vous reconnaîtront là l'acronyme de *Visual Basic Project*. Ils sont décidément très forts, chez Microsoft). Ce fichier rassemble les informations générales de votre application (en gros, la structure des différents objets Form, qui sont le squelette de toute application)

- un fichier par objet **Form** créé, fichier portant l'extension ***.frm**. Ne soyez pas si impatients, vous saurez très bientôt ce qu'est un objet Form. Toujours est-il que si votre application comporte six Form, vous aurez en plus du fichier "projet", six fichiers "Form" à sauvegarder. Chacun de ces fichiers comporte les objets contenus par la "Form", ainsi que tout le code des procédures liées à ces objets.
- éventuellement, d'autres fichiers correspondant à d'autres éléments de l'application, éléments dont nous parlerons plus tard (**modules**, **modules de classe**.

La destruction de l'un quelconque de ces fichiers vous portera naturellement un préjudice que l'on ne saurait sous-estimer.

D'autre part, je tiens à signaler dès maintenant qu'il est extrêmement périlleux de procéder à des "copier – coller" de Form, car le fichier structure (vbp) possède une tendance affirmée à se mélanger complètement les crayons en pareil cas.

Conclusion, **on crée un nouveau projet à chaque nouvelle application, et on ne déroge jamais à cette règle d'or**. Il ne doit jamais y avoir deux projets ouverts en même temps dans la même fenêtre VB, sous peine de graves représailles de la part du logiciel.

Tant que votre projet est ouvert sous cette forme d'une collection de fichiers vbp et frm, vous pouvez naturellement l'exécuter afin de le tester et de jouer au célèbre jeu des 7 777 erreurs. Lors de l'exécution, le langage est alors ce qu'on appelle «compilé à la volée». C'est-à-dire que VB traduit vos lignes de code au fur et à mesure en langage machine, puis les exécute. Cela ralentit naturellement considérablement l'exécution, même si sur de petites applications, c'est imperceptible. Mais dès que ça commence à grossir«

Voilà pourquoi, une fois l'application (le "projet") mis au point définitivement, VB vous propose de le **compiler** une bonne fois pour toutes, créant ainsi un unique fichier ***.exe**. Ce fichier contient cette fois à lui seul l'ensemble de votre projet, form, code, et tutti quanti. Et il peut naturellement être exécuté sans l'ouverture – donc la possession - préalable de Visual Basic (à un détail près, que nous réexaminerons plus loin dans ce cours).

Un projet terminé est donc un projet compilé.

Et qui, accessoirement, fonctionne sans erreurs.

haut de page

3. L'INTERFACE VB

Je ne me lancerai pas ici dans la description exhaustive de l'interface très riche (trop?) de ce langage. Vous pourrez trouver ce type de descriptif dans n'importe quel manuel du commerce. Cette partie a pour seul but de signaler les points importants.

- le langage possède un **vérificateur de syntaxe en temps réel**. C'est-à-dire que l'éditeur de code détecte les instructions non légitimes au fur et à mesure que vous les entrez, et il les signale par un texte mis en rouge, et pour faire bonne mesure, par un message d'erreur. Il est donc impossible de laisser traîner des erreurs de syntaxe en VB, ce qui je n'en doute pas, rendra les amateurs de langage C inconsolables. Qu'ils se rassurent toutefois, VB ne corrige ni les fautes de logique, ni les fautes fonctionnelles. Il reste donc tout de même de quoi se faire plaisir.

- De même, le code est immédiatement **mis en couleurs** par l'éditeur.
    - le **bleu** correspond aux mots réservés du langage (instructions, mots-clés...)
    - le **vert** correspond à un commentaire (toute ligne commençant par un guillemet simple – quote – est considérée comme un commentaire).
- tout mot reconnu par l'éditeur (nom d'objet, instruction) voit sa première lettre transformée automatiquement en **majuscule**.
- tout nom d'objet suivi d'un point voit s'afficher une **liste déroulante** contenant l'intégralité des propriétés et des méthodes disponibles pour cet objet. A contrario, cela signifie qu'un objet ne faisant pas apparaître une liste déroulante dans le code est un objet non reconnu (qui n'existe pas). De même, une instruction ne prenant pas automatiquement de majuscule initiale est une instruction non reconnue.
- il est possible de réaliser une exécution **pas à pas** via la commande appropriée du menu.
- il est possible d'insérer **des points d'arrêt** pour faciliter le déboguage.
- dans le cas d'un pas à pas comme d'un point d'arrêt, il est possible de connaître la valeur actuelle d'une variable en **pointant** (sans cliquer !) **la souris** sur une occurrence de cette variable dans le code.

Fenêtre d'une application VB en exécution Pas à Pas